下载APP

02 | NumPy（上）：核心数据结构详解

2021-10-13 方远

《PyTorch深度学习实战》 课程介绍 >

讲述：方远
时长 17:41 大小 16.20M

你好，我是方远。

通过前面两节课，我们已经对 PyTorch 有了初步的了解，你是不是迫不及待想要动手玩转 PyTorch 了？先别着急，我们有必要先品尝一道“前菜”，它就是 NumPy。

为什么我们要先拿下 NumPy 呢？我相信，无论你正在从事或打算入门机器学习，不接触 NumPy 几乎不可能。现在的主流深度学习框架 PyTorch 与 Tensorflow 中最基本的计算单元 Tensor，都与 NumPy 数组有着类似的计算逻辑，可以说掌握了 NumPy 对学习这两种框架都有很大帮助。

另外，NumPy 还被广泛用在 Pandas，SciPy 等其他数据科学与科学计算的 Python 模块中。而我们日常用得越来越多的人脸识别技术（属于计算机视觉领域），其原理本质上就

是先把图片转换成 NumPy 的数组，然后再进行一系列处理。

为了让你真正吃透 NumPy，我会用两节课的内容讲解 NumPy。这节课，我们先介绍 NumPy 的数组、数组的关键属性以及非常重要的轴的概念。

什么是 NumPy

NumPy 是用于 Python 中科学计算的一个基础包。它提供了一个多维度的数组对象（稍后展开），以及针对数组对象的各种快速操作，例如排序、变换，选择等。NumPy 的安装方式非常简单，可以使用 Conda 安装，命令如下：

```
1 conda install numpy
```

复制代码

或使用 pip 进行安装，命令如下：

```
1 pip install numpy
```

复制代码

NumPy 数组

刚才所说的数组对象是 NumPy 中最核心的组成部分，这个数组叫做 ndarray，是“N-dimensional array”的缩写。其中的 N 是一个数字，指代维度，例如你常常能听到的 1-D 数组、2-D 数组或者更高维度的数组。

在 NumPy 中，数组是由 numpy.ndarray 类来实现的，它是 NumPy 的核心数据结构。我们今天的内容就是围绕它进行展开的。

学习一个新知识，我们常用的方法就是跟熟悉的东西做对比。NumPy 数组从逻辑上来看，与其他编程语言中的数组是一样的，索引也是从 0 开始。而 Python 中的列表，其实也可以达到与 NumPy 数组相同的功能，但它们又有差异，做个对比你就能体会到 NumPy 数组的特点了。

1.Python 中的列表可以动态地改变，而 NumPy 数组是不可以的，它在创建时就有固定大小了。改变 Numpy 数组长度的话，会新创建一个新的数组并且删除原数组。

2.NumPy 数组中的数据类型必须是一样的，而列表中的元素可以是多样的。

3.NumPy 针对 NumPy 数组一系列的运算进行了优化，使得其速度特别快，并且相对于 Python 中的列表，同等操作只需使用更少的内存。

创建数组

好，那就让我们来看看 NumPy 数组是怎么创建的吧？

最简单的方法就是把一个列表传入到 np.array() 或 np.asarray() 中，这个列表可以是任意维度的。np.array() 属于深拷贝，np.asarray() 则是浅拷贝，它们的区别我们下节课再细讲，这里你有个印象就行。

我们可以先试着创建一个一维的数组，代码如下。

复制代码

```
>>>import numpy as np
>>>#引入一次即可

>>>arr_1_d = np.asarray([1])
>>>print(arr_1_d)
[1]
```

再创建一个二维数组：

复制代码

```
>>>arr_2_d = np.asarray([[1, 2], [3, 4]])
>>>print(arr_2_d)
[[1 2]
 [3 4]]
```

你也可以试试自己创建更高维度的数组。

数组的属性

作为一个数组，NumPy 有一些固有的属性，我们今天来介绍非常常用且关键的数组维度、形状、size 与数据类型。

ndim

ndim 表示数组维度（或轴）的个数。刚才创建的数组 arr_1_d 的轴的个数就是 1，arr_2_d 的轴的个数就是 2。

复制代码

```
>>>arr_1_d.ndim
1
>>>arr_2_d.ndim
2
```

shape

shape 表示数组的维度或形状， 是一个整数的元组，元组的长度等于 ndim。

arr_1_d 的形状就是（1，）（一个向量）， arr_2_d 的形状就是 (2, 2)（一个矩阵）。

复制代码

```
>>>arr_1_d.shape
(1,)
>>>arr_2_d.shape
(2, 2)
```

shape 这个属性在实际中用途还是非常广的。比如说，我们现在有这样的数据 (B, W, H, C)，熟悉深度学习的同学肯定会知道，这代表一个 batch size 为 B 的（W，H，C）数据。

现在我们需要根据（W，H，C）对数据进行变形或者其他处理，这时我们可以直接使用 input_data.shape[1:3]获取到数据的形状，而不需要直接在程序中硬编程、直接写好输入数据的宽高以及通道数。

在实际的工作当中，我们经常需要对数组的形状进行变换，就可以使用 arr.reshape() 函数，在不改变数组元素内容的情况下变换数组的形状。但是你需要注意的是，**变换前与变**

换后数组的元素个数需要是一样的，请看下面的代码。

复制代码

```
>>>arr_2_d.shape
(2, 2)
>>>arr_2_d
[[1 2]
 [3 4]]
# 将arr_2_d reshape为(4，1)的数组
>>>arr_2_d.reshape((4，1))
array([[1],
       [2],
       [3],
       [4]])
```

我们还可以使用 np.reshape(a, newshape, order) 对数组 a 进行 reshape，新的形状在 newshape 中指定。

这里需要注意的是，reshape 函数有个 **order 参数**，它是指以什么样的顺序读写元素，其中有这样几个参数。

‘C’ ：默认参数，使用类似 C-like 语言（行优先）中的索引方式进行读写。

‘F’ ：使用类似 Fortran-like 语言（列优先）中的索引方式进行读写。

‘A’ ：原数组如果是按照 ‘C’ 的方式存储数组，则用 ‘C’ 的索引对数组进行 reshape，否则使用’ F’ 的索引方式。

reshape 的过程你可以这样理解，首先需要根据指定的方式（ ‘C’ 或’ F’ ）将原数组展开，然后再根据指定的方式写入到新的数组中。

这是什么意思呢？先看一个简单的 2 维数组的例子。

复制代码

```
>>>a = np.arange(6).reshape(2,3)
array([[0, 1, 2],
       [3, 4, 5]])
```

我们要将数组 a，按照’ C’ 的方式 reshape 成 (3,2)，可以这样操作。首先将原数组展开，对于 ‘C’ 的方式来说是行优先，最后一个维度最优先改变，所以展开结果如下，序号那一列代表展开顺序。

序号	坐标	数值
1	(0,0)	0
2	(0,1)	1
3	(0,2)	2
4	(1,0)	3
5	(1,1)	4
6	(1,2)	5

极客时间

所以，reshape 后的数组，是按照 0，1，2，3，4，5 这个序列进行写入数据的。reshape 后的数组如下表所示，序号代表写入顺序。

序号	坐标	数值
1	(0,0)	0
2	(0,1)	1
3	(1,0)	2
4	(1,1)	3
5	(2,0)	4
6	(2,1)	5

极客时间

接下来，再看看将数组 a，按照’ F’ 的方式 reshape 成 (3,2) 要如何处理。

对于行优先的方式，我们应该是比较熟悉的，而‘F’方式是列优先的方式，这一点对于没有使用过列优先的同学来说，可能比较难理解一点。

首先是按列优先展开原数组，列优先意味着最先变化的是数组的第一个维度。下表是展开后的结果，序号是展开顺序，这里请注意下**坐标的变换方式**（第一个维度最先变化）。

序号	坐标	数值
1	(0,0)	0
2	(1,0)	3
3	(0,1)	1
4	(1,1)	4
5	(0,2)	2
6	(1,2)	5

极客时间

所以，reshape 后的数组，是按照 0，3，1，4，2，5 这个序列进行写入数据的。reshape 后的数组如下表所示，序号代表写入顺序，为了显示直观，我将相同行以同样颜色显示了。

序号	坐标	数值
1	(0,0)	0
2	(1,0)	3
3	(2,0)	1
4	(0,1)	4
5	(1,1)	2
6	(2,1)	5

极客时间

这里我给你留一个小练习，你可以试试对多维数组的 reshape 吗？

不过，大部分时候还是使用’ C’ 的方式比较多，也就是行优先的形式。至少目前为止我还没有使用过’ F’ 与’ A’ 的方式。

size

size，也就是数组元素的总数，它就等于 shape 属性中元素的乘积。

请看下面的代码，arr_2_d 的 size 是 4。

复制代码

```
>>>arr_2_d.size
4
```

dtype

最后要说的是 dtype，它是一个描述数组中元素类型的对象。使用 dtype 属性可以查看数组所属的数据类型。

NumPy 中大部分常见的数据类型都是支持的，例如 int8、int16、int32、float32、float64 等。dtype 是一个常见的属性，在创建数组，数据类型转换时都可以看到它。

首先我们看看 arr_2_d 的数据类型：

复制代码

```
>>>arr_2_d.dtype
dtype('int64')
```

你可以回头看一下刚才创建 arr_2_d 的时候，我们并没有指定数据类型，如果没有指定数据类型，NumPy 会自动进行判断，然后给一个默认的数据类型。

我们再看下面的代码，我们在创建 arr_2_d 时，对数据类型进行了指定。

复制代码

```
>>>arr_2_d = np.asarray([[1, 2], [3, 4]], dtype='float')
>>>arr_2_d.dtype
dtype('float64')
```

数组的数据类型当然也可以改变，我们可以使用 astype() 改变数组的数据类型，不过改变数据类型会创建一个新的数组，而不是改变原数组的数据类型。

请看后面的代码。

复制代码

```
>>>arr_2_d.dtype
dtype('float64')
>>>arr_2_d.astype('int32')
array([[1, 2],
       [3, 4]], dtype=int32)
>>>arr_2_d.dtype
dtype('float64')
# 原数组的数据类型并没有改变
>>>arr_2_d_int = arr_2_d.astype('int32')
>>>arr_2_d_int.dtype
dtype('int32')
```

但是，我想提醒你，**不能通过直接修改数据类型来修改数组的数据类型**，这样代码虽然不会报错，但是数据会发生改变，请看下面的代码：

复制代码

```
>>>arr_2_d.dtype
dtype('float64')
>>>arr_2_d.size
4
>>>arr_2_d.dtype='int32'
>>>arr_2_d
array([[         0, 1072693248,          0, 1073741824],
       [         0, 1074266112,          0, 1074790400]], dtype=int32)
```

1 个 float64 相当于 2 个 int32，所以原有的 4 个 float32，会变为 8 个 int32，然后直接输出这个 8 个 int32。

其他创建数组的方式

除了使用 np.asarray 或 np.array 来创建一个数组之外，NumPy 还提供了一些按照既定方式来创建数组的方法，我们只需按照要求，提供一些必要的参数即可。

np.ones() 与 np.zeros()

np.ones() 用来创建一个全 1 的数组，必须参数是指定数组的形状，可选参数是数组的数据类型，你可以结合下面的代码进行理解。

复制代码

```
>>>np.ones()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: ones() takes at least 1 argument (0 given)
# 报错原因是没有给定形状的参数
>>>np.ones(shape=(2,3))
array([[1., 1., 1.],
       [1., 1., 1.]])
>>>np.ones(shape=(2,3), dtype='int32')
array([[1, 1, 1],
       [1, 1, 1]], dtype=int32)
```

创建全 0 的数组是 np.zeros()，用法与 np.ones() 类似，我们就不举例了。

那这两个函数一般什么时候用呢？例如，如果需要初始化一些权重的时候就可以用上，比如说生成一个 2x3 维的数组，每个数值都是 0.5，可以这样做。

复制代码

```
>>>np.ones((2, 3)) * 0.5
array([[0.5, 0.5, 0.5],
       [0.5, 0.5, 0.5]]
```

np.arange()

我们还可以使用 np.arange([start,]stop, [step,]dtype=None) 创建一个在[start, stop)区间的数组，元素之间的跨度是 step。

start 是可选参数，默认为 0。stop 是必须参数，区间的终点，请注意，刚才说的区间是一个**左闭右开区间**，所以数组并不包含 stop。step 是可选参数，默认是 1。

复制代码

```
# 创建从0到4的数组
>>>np.arange(5)
array([0, 1, 2, 3, 4])
# 从2开始到4的数组
>>>np.arange(2, 5)
array([2, 3, 4])
# 从2开始，到8的数组，跨度是3
>>>np.arange(2, 9, 3)
array([2, 5, 8])
```

np.linspace()

最后，我们也可以用 np.linspace（start, stop, num=50, endpoint=True, retstep=False, dtype=None）创建一个数组，具体就是创建一个从开始数值到结束数值的等差数列。

start：必须参数，序列的起始值。

stop：必须参数，序列的终点。

num：序列中元素的个数，默认是 50。

endpoint：默认为 True，如果为 True，则数组最后一个元素是 stop。

retstep：默认为 False，如果为 True，则返回数组与公差。

复制代码

```
# 从2到10，步长为3的等差数列
>>>np.linspace(start=2, stop=10, num=3)
```

np.arange 与 np.linspace 也是比较常见的函数，比如你要作图的时候，可以用它们生成 x 轴的坐标。例如，我要生成一个 $y = x^2$ 的图片，x 轴可以用 np.linespace() 来生成。

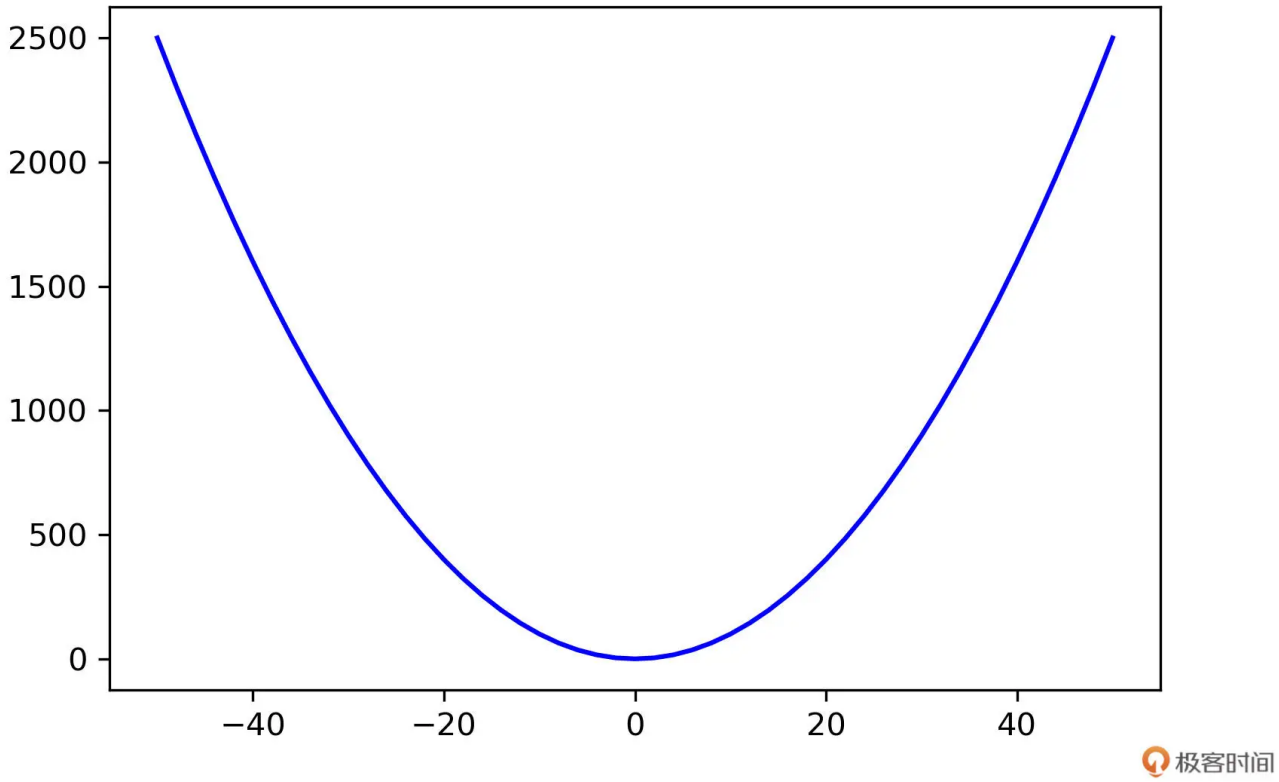
复制代码

```
import numpy as np
import matplotlib.pyplot as plt

X = np.arange(-50, 51, 2)
Y = X ** 2

plt.plot(X, Y, color='blue')
```

```
plt.legend()
plt.show()
```



数组的轴

这是一个非常重要的概念，也是 NumPy 数组中最不好理解的一个概念。它经常出现在 np.sum()、np.max() 这样关键的聚合函数中。

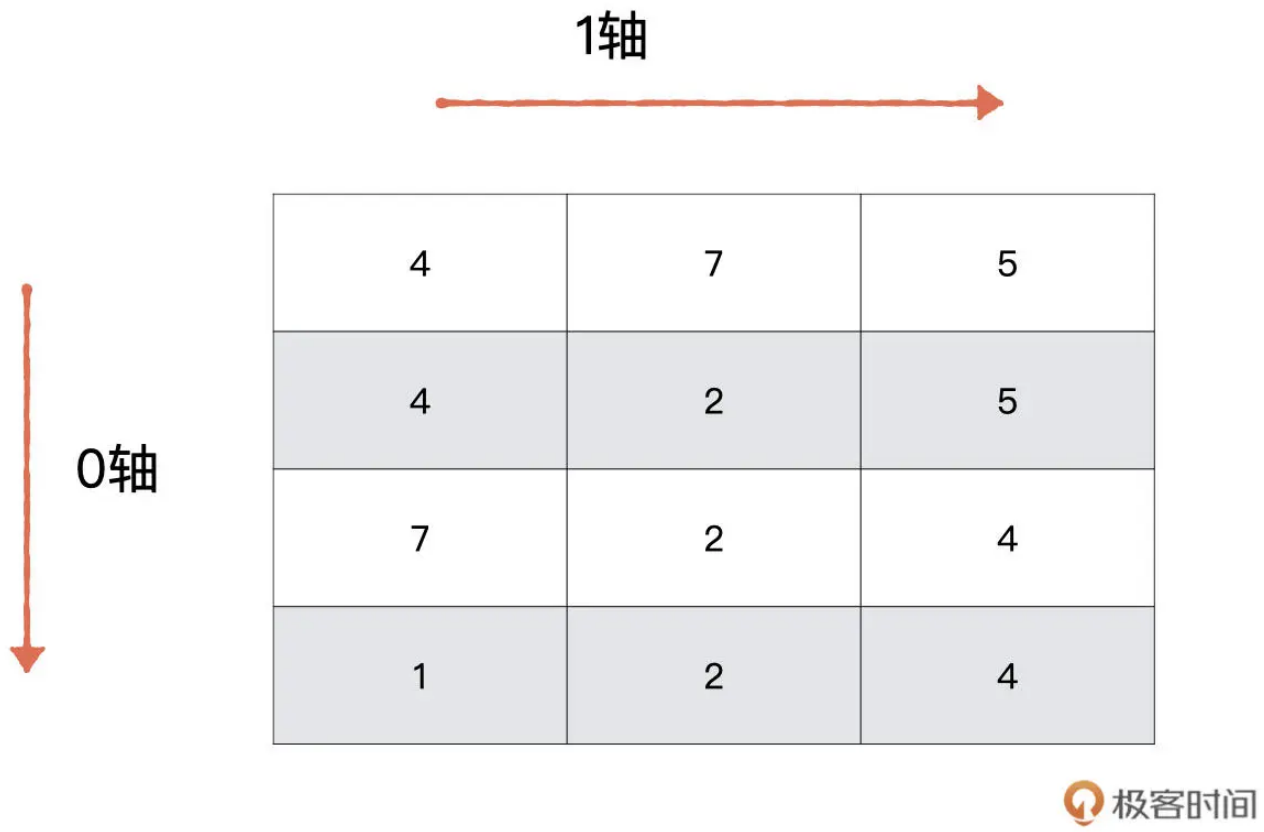
我们用这样一个问题引出，同一个函数如何根据轴的不同来获得不同的计算结果呢？比如现在有一个 (4,3) 的矩阵，存放着 4 名同学关于 3 款游戏的评分数据。

复制代码

```
>>>interest_score = np.random.randint(10, size=(4, 3))
>>>interest_score
array([[4, 7, 5],
       [4, 2, 5],
       [7, 2, 4],
       [1, 2, 4]])
```

第一个需求是，计算每一款游戏的评分总和。这个问题如何解决呢，我们一起分析一下。

数组的轴即数组的维度，它是从 0 开始的。对于我们这个二维数组来说，有两个轴，分别是代表行的 0 轴与代表列的 1 轴。如下图所示。

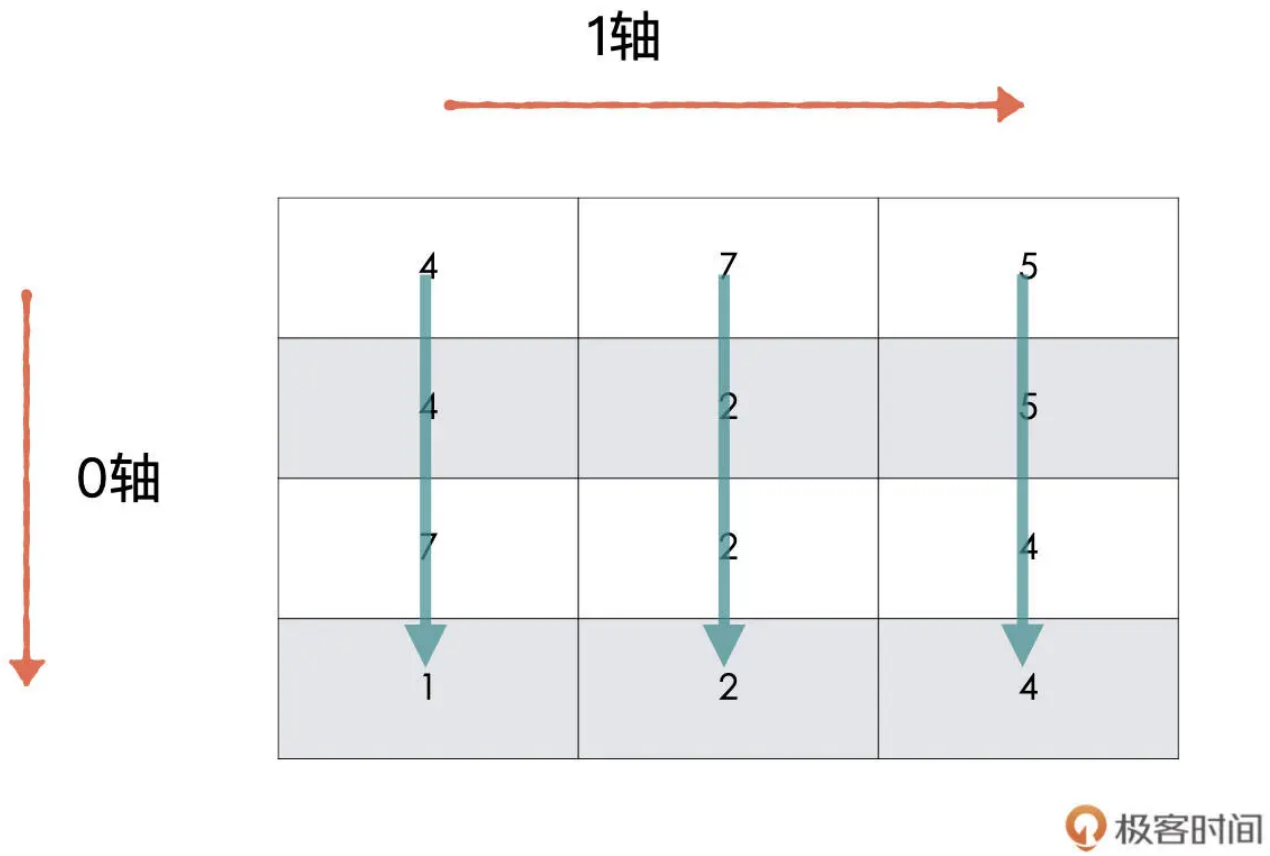


我们的问题是要计算每一款游戏的评分总和，也就是沿着 0 轴的方向进行求和。所以，我们只需要在求和函数中指定沿着 0 轴的方向求和即可。

复制代码

```
>>> np.sum(interest_score, axis=0)
array([16, 13, 18])
```

计算方向如绿色箭头所示：

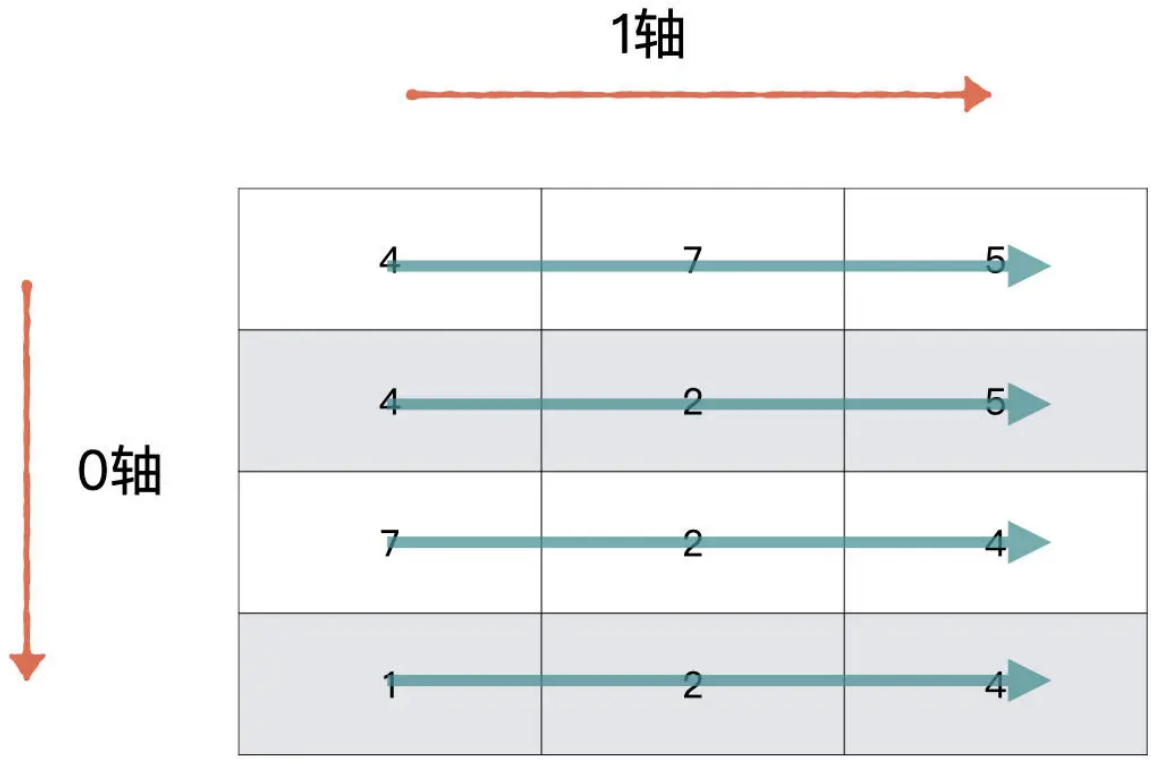


第二个问题是要计算每名同学的评分总和，也就是要沿着 1 轴方向对二维数组进行操作。所以，我们只需要将 axis 参数设定为 1 即可。

复制代码

```
>>> np.sum(interest_score, axis=1)
array([16, 11, 13,  7])
```

计算方向如绿色箭头所示。



极客时间

二维数组还是比较好理解的，那多维数据该怎么办呢？你有没有发现，其实当 axis=i 时，就是按照第 i 个轴的方向进行计算的，或者可以理解为第 i 个轴的数据将会被折叠或聚合到一起。

形状为 (a, b, c) 的数组，沿着 0 轴聚合后，形状变为 (b, c)；沿着 1 轴聚合后，形状变为 (a, c)；

沿着 2 轴聚合后，形状变为 (a, b)；更高维数组以此类推。

接下来，我们再看一个多维数组的例子。对数组 a，求不同维度上的最大值。

复制代码

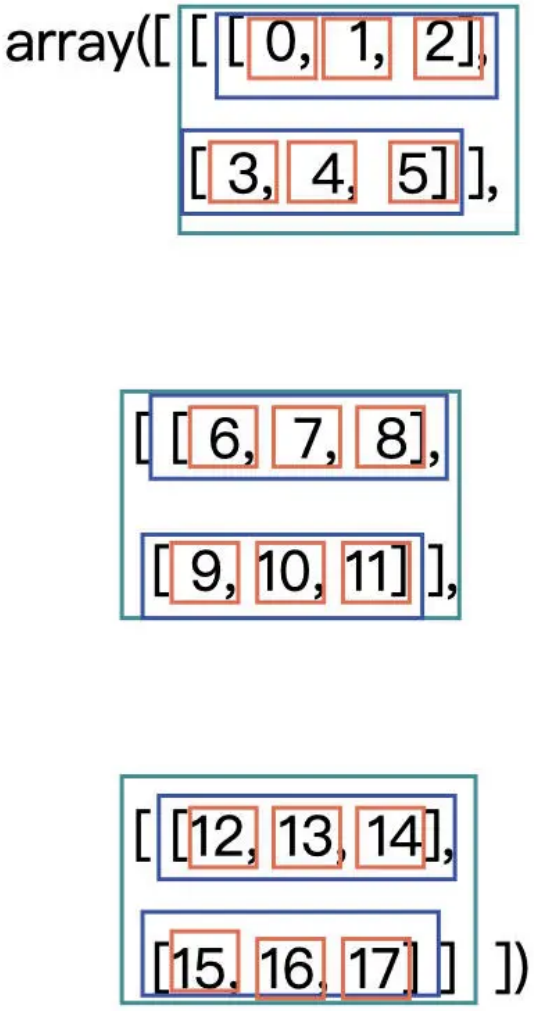
```
>>> a = np.arange(18).reshape(3,2,3)
>>> a
array([[[ 0,  1,  2],
        [ 3,  4,  5]],

       [[ 6,  7,  8],
        [ 9, 10, 11]],

       [[12, 13, 14],
        [15, 16, 17]]])
```

我们可以将同一个轴上的数据看做同一个单位，那聚合的时候，我们只需要在同级别的单位上进行聚合就可以了。

如下图所示，绿框代表沿着 0 轴方向的单位，蓝框代表着沿着 1 轴方向的单位，红框代表着 2 轴方向的单位。



极客时间

当 axis=0 时，就意味着将三个绿框的数据聚合在一起，结果是一个（2，3）的数组，数组内容为：

$$\begin{bmatrix} [(max(a_{000}, a_{100}, a_{200}), max(a_{001}, a_{101}, a_{201}), max(a_{002}, a_{102}, a_{202}))], \\ [(max(a_{010}, a_{110}, a_{210}), max(a_{011}, a_{111}, a_{211}), max(a_{012}, a_{112}, a_{212}))] \end{bmatrix}$$

代码如下：

```
>>> a.max(axis=0)
array([[12, 13, 14],
       [15, 16, 17]])
```

当 axis=1 时，就意味着每个绿框内的蓝框聚合在一起，结果是一个（3，3）的数组，数组内容为：

$$[\ [(max(a_{000}, a_{010}), max(a_{001}, a_{011}), max(a_{002}, a_{012}))],\\ [(max(a_{100}, a_{110}), max(a_{101}, a_{111}), max(a_{102}, a_{112}))],\\ [(max(a_{200}, a_{210}), max(a_{201}, a_{211}), max(a_{202}, a_{212}))],\ ]$$

代码如下：

```
>>> a.max(axis=1)
array([[ 3,  4,  5],
       [ 9, 10, 11],
       [15, 16, 17]])
```

当 axis=2 时，就意味着每个绿框中的红框聚合在一起，结果是一个（3，2）的数组，数组内容如下所示：

$$[\ [(max(a_{000}, a_{001}, a_{002}), max(a_{010}, a_{011}, a_{012}))],\\ [(max(a_{100}, a_{101}, a_{102}), max(a_{110}, a_{111}, a_{112}))],\\ [(max(a_{200}, a_{201}, a_{202}), max(a_{210}, a_{211}, a_{212}))],\ ]$$

代码如下：

```
>>> a.max(axis=2)
array([[ 2,  5],
       [ 8, 11],
```

```
4         [14, 17]])
```

axis 参数非常常见，不光光出现在刚才介绍的 sum 与 max，还有很多其他的聚合函数也会用到，例如 min、mean、argmin（求最小值下标）、argmax（求最大值下标）等。

小结

恭喜你完成了这节课的学习。其实你只要有一些其他语言的编程基础，学 Numpy 还是非常容易的。这里我想再次强调一下为什么 NumPy 这道前菜必不可少。

其实 Numpy 的很多知识点是与 PyTorch 融会贯通的，例如 PyTorch 中的 Tensor。而且 Numpy 在机器学习中常常被用到，很多模块都要基于 NumPy 展开，尤其是在数据的预处理和膜后处理中。

NumPy 是用于 Python 中科学计算的一个基础包。它提供了一个多维度的数组对象，以及针对数组对象的各种快速操作。为了让你有更直观的体验，我们学习了创建数组的四种方式。

其中你重点要掌握的方法，就是如何使用 **np.asarray** 创建一个数组。这里涉及数组属性（ndim、shape、dtype、size）的灵活使用，特别是数组的形状变化与数据类型转换。

最后，我为你介绍了数组轴的概念，我们需要在数组的聚合函数中灵活运用它。虽然这个概念十分常用，但却不好理解，建议你根据我课程里的例子仔细揣摩一下，从 2 维数组一步步推理到多维数组，根据轴的不同，数组聚合的方向是如何变化的。

下一节课，我们要继续学习 NumPy 中常用且重要的功能。

每课一练

在刚才用户对游戏评分的那个问题中，你能计算一下每位用户对三款游戏的打分的平均分吗？

欢迎你在留言区记录你的疑问或者收获，也推荐你把这节课分享给你的朋友。

分享给需要的人，Ta订阅后你可得 20 元现金奖励

生成海报并分享

赞 0 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 01 | PyTorch：网红中的顶流明星

下一篇 03 | NumPy（下）：深度学习中的常用操作

精选留言 (6)

写留言

jssfy
2021-10-13

pytorch哪些场景必须或者强烈建议用np? 不用python数组的原因是什么?可否举个例子?

作者回复: 你好，jssfy。首先谢谢你的留言。
pytorch是用于深度学习的框架，如果涉及到数组使用都强烈建议使用np，原因有以下几点：
1. np提供了丰富的矩阵运算，非常适合深度学习处理数据时使用。
例如，数组中某维度的数据拼接，在实际训练场景中，可用于增加训练数据的特征维度等，np中一个函数即可搞定。
再例如，np中的argmax函数，可以让我们在做分类任务时轻松取到最大概率的分类是什么。
等等这些np数组中强大的功能，都是python数组所不具备的。
2. np节约内存
np数组对内存的优化很好，长度越大，其相比python的数组占用内存越少。

5

木木
2021-10-14

老师好。老师讲的最后一个三维的数组聚合，我看的不是特别明白。聚合是求最大值吗？

作者回复: 你好，木木。感谢你的留言，聚合函数是一类函数的统称。
例如，求最大值，最小值，求均值等等，都可以叫做聚合函数。
如果还有哪里不明白，可以进入到读者交流群，再继续沟通^^

大大大大大大黄同学

2021-10-14

方老师，您好。请问按照'F'对numpy数组变形为(3,2)是否可以理解为：先对原始数组arr进行转置得到arr_T，按照'C'对arr_T变形为(2,3)，再转置回来。

展开

作者回复: 你好，大大大大大大黄同学，谢谢你的留言并且为你提出自己的见解点赞。
不过这样理解稍微有点不正确。
首先是矩阵转置是逻辑概念，而从'C'与'F'的方式进行reshape是物理层面的。
其次，这种思考方式，如果数组变成高维度了该怎么办呢？
最后是，文章里是从(2,3)变为(3,2)，(2,3)的数组转置后已经是(3,2)了。：）

lwg0452

2021-10-13

思考题
np.average(interest_score, axis=1)

作者回复: 你好，lwg0452。感谢你的留言^^。是的，这么做是正确的👍。
也可以用np.mean。对于普通求平均来说，他们都是一样的，也就是说np.average(interest_score, axis=1)与np.mean(interest_score, axis=1)的输出是一样的。
不过np.average还可以用于加权平均，比如给定每个游戏的权值weights=np.array([4,2,7])，计算每名同学对三款游戏的加权平均就可以这样做，
np.average(interest_score, weights=weights, axis=1)。
加权平均计算方式的话，用第一名同学举例就是
interest_score[0][0] * weights[0]/weights.sum() + interest_score[0][1] * weights[1]/weights.sum() + interest_score[0][2] * weights[0]/weights.sum()

人间失格

2021-10-13

给老师点赞，老师轴抽象的很好，比我初次看更具体，希望后面网络结构和优化也能这样。请问numpy快和并行的原因是因为底层storage方式吗？

展开

作者回复: 你好，人间失格。谢谢你的留言也感谢你的认可。：）我会用通俗易通的言语来讲解专栏的内容。

Numpy快是因为底层是用C、C++和Fortran写的，并在底层对各种运算进行了高度的优化。

AstrHan

2021-10-13

数组轴的这个例子好记！老师是自行车爱好者啊，每张头图都是，，

作者回复: 你好，AstrHan，谢谢你的留言。谢谢你的认可。是啊，自行车爱好者：）。